Algebraic iterative reconstruction in the CCPi Core Imaging Library (CIL)

Jakob Sauer Jørgensen – DTU

Gemma Fardell – STFC

Laura Murgatroyd – STFC

Evangelos Papoutsellis – STFC

Edoardo Pasca – STFC

Before we start: Week 1 feedback

https://tinyurl.com/SC4CT1

- *“During the work with the micro projects, a lot of students wanted to discuss ideas with Jakob. It could be nice to make help waitlist on the blackboard.”* → Will do today!
- *“I didn't learn much from the CIL exercises, as the servers were not very stable, and they weren't able to load the data.”* → New servers today!
- *“Please allow the people who have attended this course to use the jupyter notebook even after the course ends..... currently the acess is only untill the course end date.”* → Will see what I can do!

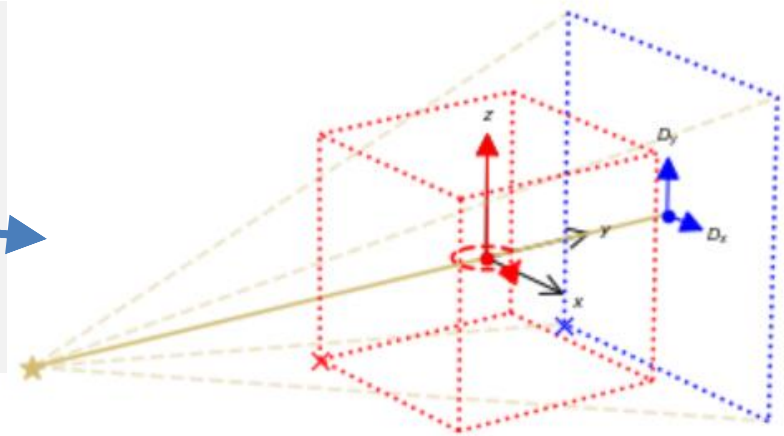
What is the Core Imaging Library?

- A Python library for processing and reconstruction of tomography data.
- Special emphasis on ”challenging data sets”: noisy, non-standard, incomplete, multi-channel, ...
- Optimised standard methods: FBP, FDK
- Highly modular to allow creation of bespoke pipelines.
- Range of **iterative reconstruction methods** and building blocks allowing users to create new ones.
- Fully open source under permissive Apache 2 license.
- Actively developed on GitHub: https://github.com/TomographicImaging/CIL

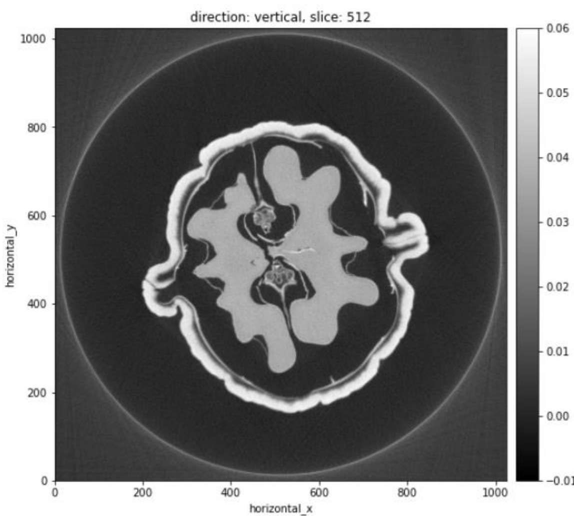
CIL
CORE IMAGING LIBRARY

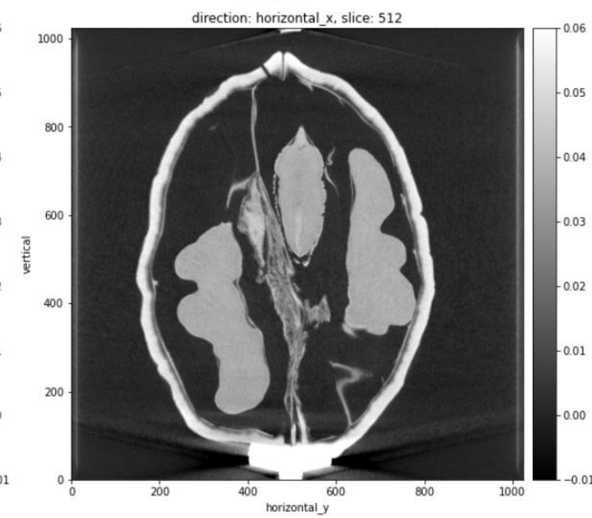
Example CIL code: Cone-beam FBP

```
data = ZEISSDataReader(filename).read()

data = TransmissionAbsorptionConverter()(data)

show_geometry(data.geometry)

recon = FDK(data).run()

show2D(recon)
```



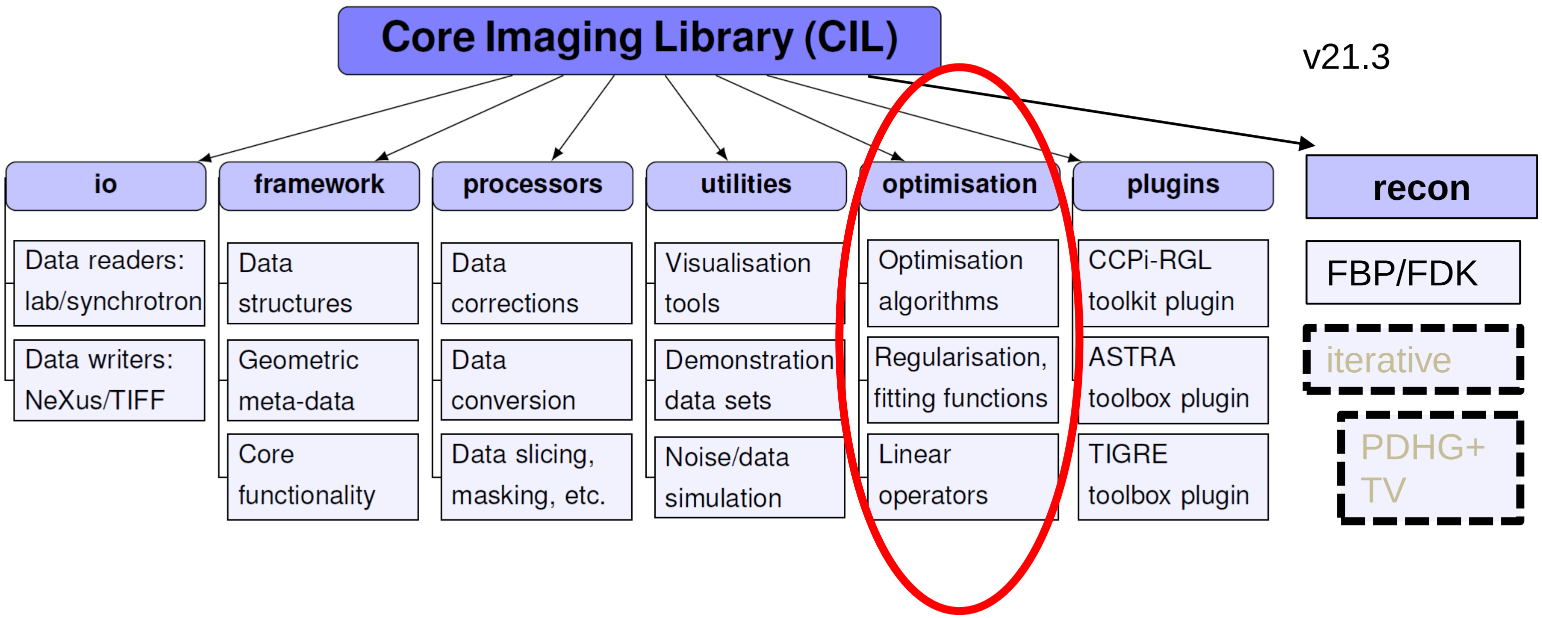
- Data readers/writers
- Pre-processing tools
- TIGRE and ASTRA backend
- 2D, 3D and 4D data
- *Near math* optimisation syntax
- Visualisation





ccpi.ac.uk/CIL

Module organization and contents



Jørgensen et al. 2021: *Core Imaging Library - Part I: a versatile Python framework for tomographic imaging,* Phil. Trans. R. Soc. A, **379**, 20200192: https://doi.org/10.1098/rsta.2020.0192

Imaging model for iterative reconstruction

Beer-Lambert for ith ray (along line L_i):

$$\int_{L_i} \mu \, ds = -\log \frac{I_i}{I_0} = b_i$$

Assume object constant in each pixel:

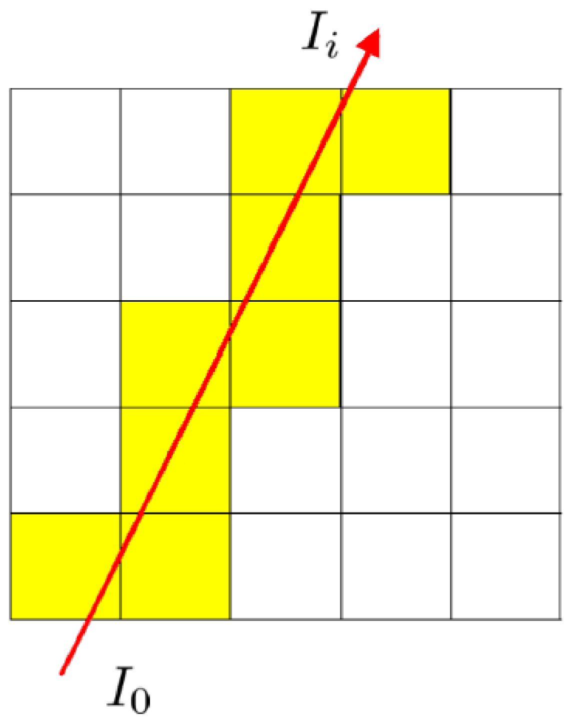
- x_j is the jth pixel value.
- a_{ij} is path length through jth pixel.

Approximate line integral by sum:

$$\sum_j a_{ij} x_j = b_i$$

Extremely large set of linear equations:

$$Ax = b$$



Operator *A*:

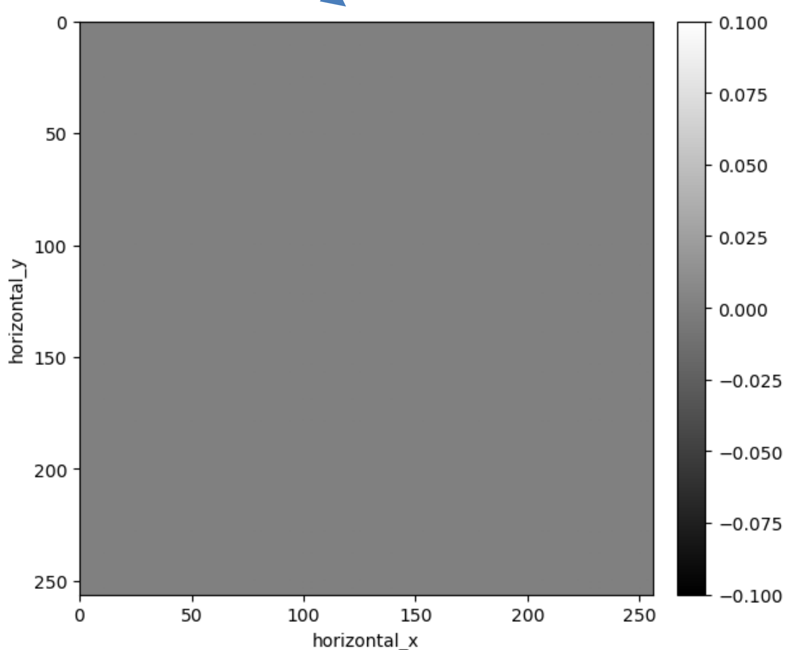
- Direct *Ax*: Projection
- Adjoint $A^T b$: Backprojection

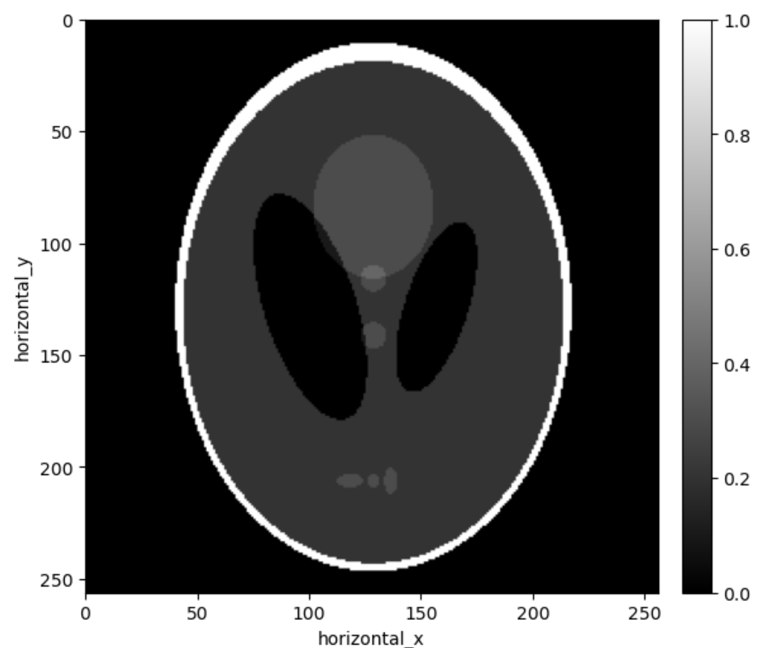
CIL ImageGeometry

```
>>> n_pixels = 256

>>> ig = ImageGeometry(voxel_num_x=n_pixels,
                       voxel_num_y=n_pixels,
                       voxel_size_x=1/n_pixels,
                       voxel_size_y=1/n_pixels)

>>> print(ig)
      Number of channels: 1
      channel_spacing: 1.0
      voxel_num : x256,y256
      voxel_size : x0.00390625,y0.00390625
      center : x0,y0
```

CIL ImageGeometry

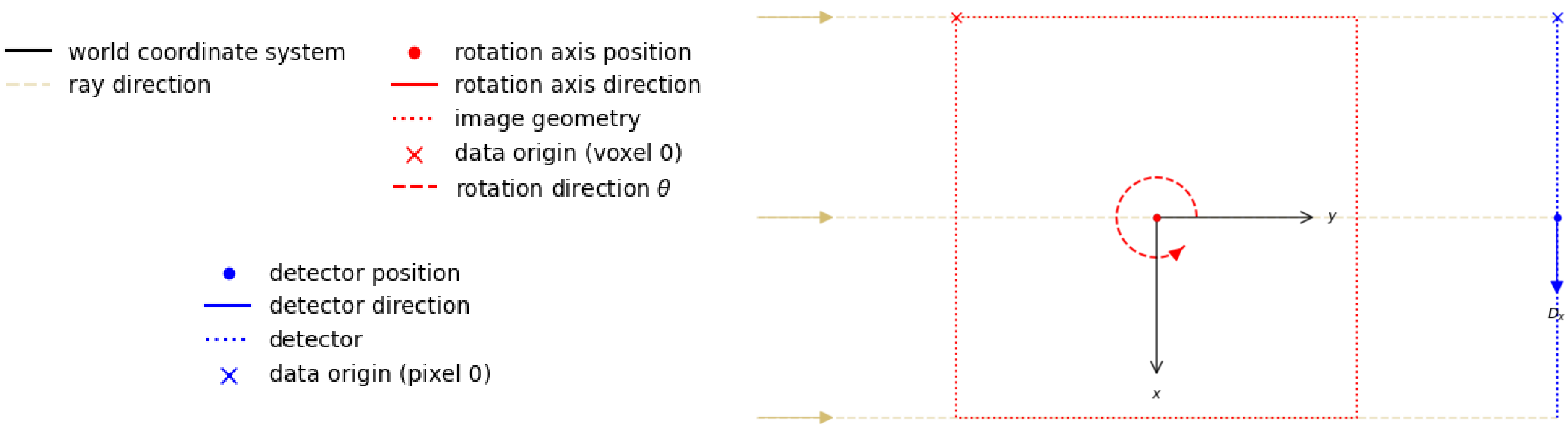
```
phantom = TomoPhantom.get_ImageData(num_model=1,
                                    geometry=ig)
show2D(phantom, origin='upper-left')

x0 = ig.allocate(0.0)
show2D(x0)
```





CIL AcquisitionGeometry

```
angles = np.linspace(0, 180, 256, endpoint=False)

ag = AcquisitionGeometry.create_Parallel2D(
            detector_position=[0,1])\
            .set_angles(angles)\
            .set_panel(n_pixels,
                       pixel_size=1/n_pixels)
show_geometry(ag)
```



CIL AcquisitionGeometry

```
>>> print(ag)
2D Parallel-beam tomography
System configuration:
        Ray direction: [0., 1.]
        Rotation axis position: [0., 0.]
        Detector position: [0., 1.]
        Detector direction x: [1., 0.]
Panel configuration:
        Number of pixels: [256   1]
        Pixel size: [0.00390625 0.00390625]
        Pixel origin: bottom-left
Channel configuration:
        Number of channels: 1
Acquisition description:
        Number of positions: 256
        Angles 0-20 in degrees:
[ 0.      ,  0.703125,  1.40625 ,  2.109375,  2.8125  , 3.515625,
  4.21875 ,  4.921875,  5.625   ,  6.328125,  7.03125 ,  7.734375,
  8.4375  ,  9.140625,  9.84375 , 10.546875, 11.25    , 11.953125,
 12.65625 , 13.359375]
```

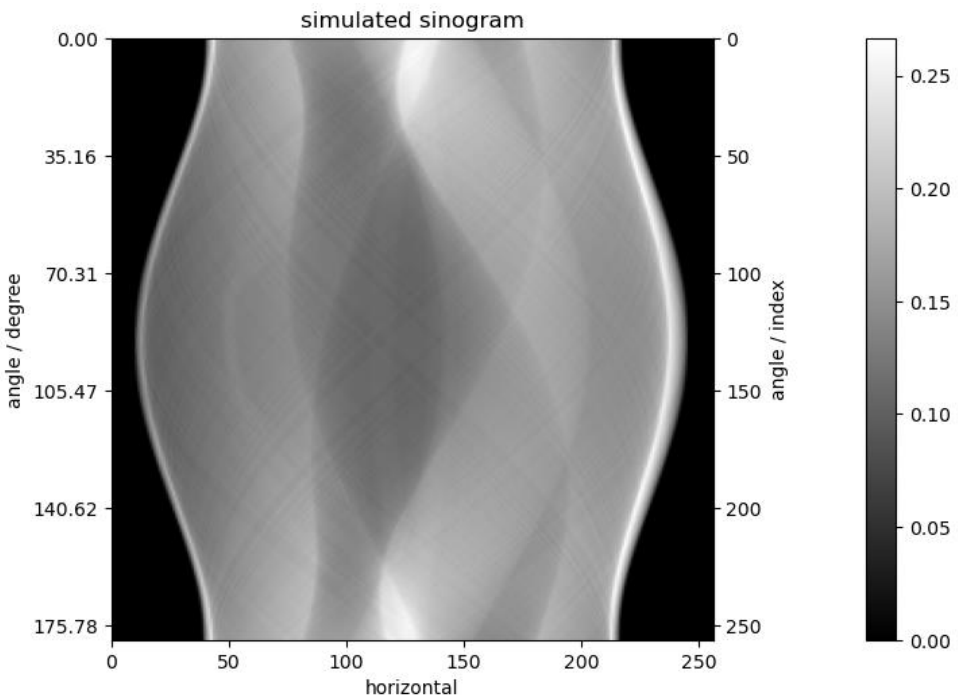
CIL AcquisitionGeometry

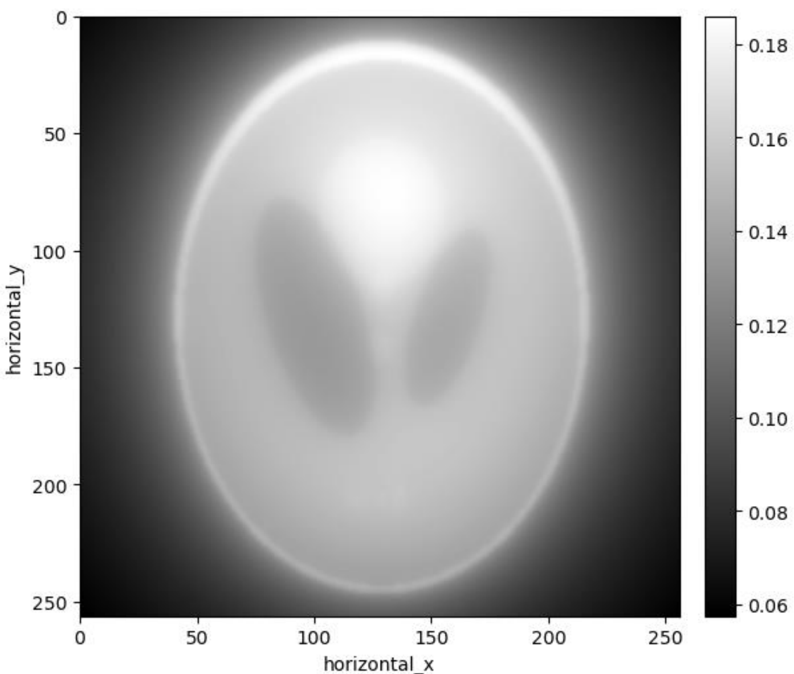
```
>>> ag.dimension_labels
      ('angle', 'horizontal')

>>> ig = ag.get_ImageGeometry()
>>> print(ig)
         Number of channels: 1
         channel_spacing: 1.0
         voxel_num : x256,y256
         voxel_size : x0.00390625,y0.00390625
         center : x0,y0

>>> ag2D = ag3D.get_centre_slice()
```

Parallel2D, Cone2D, Parallel3D, Cone3D supported, more to come.

CIL ProjectionOperator

```
>>> A = ProjectionOperator(ig, ag, "gpu")

>>> sino = A.direct(phantom)

>>> bp_image = A.adjoint(sino)
```





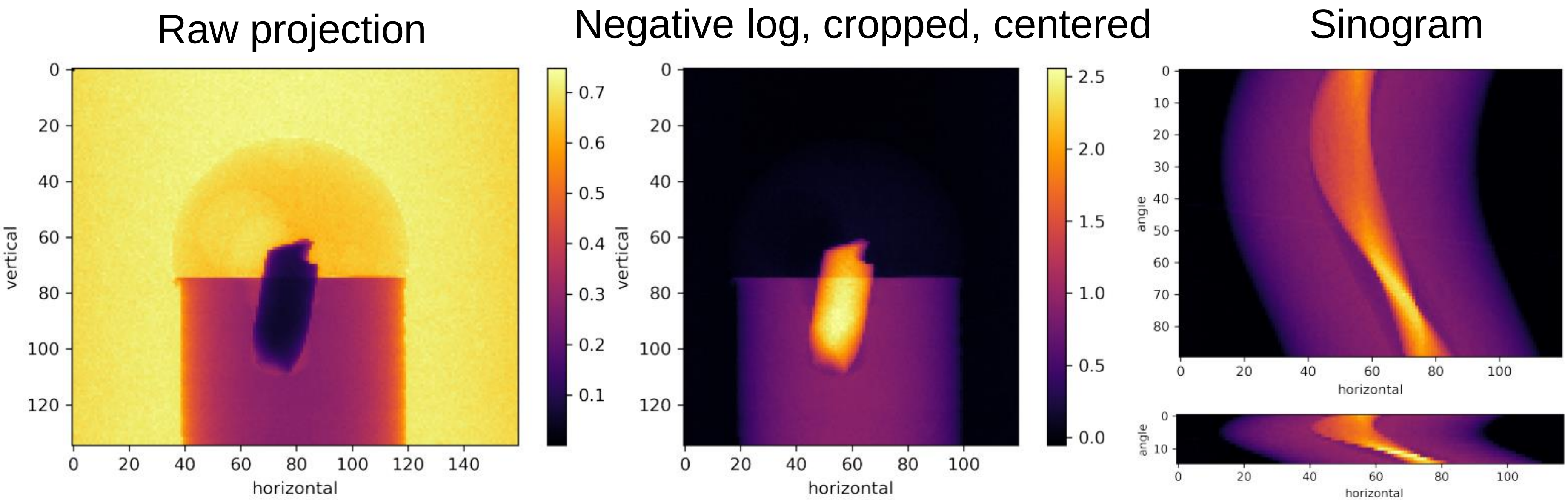
CIL ProjectionOperator

```
>>> A.domain_geometry()
      cil.framework.framework.ImageGeometry

>>> A.range_geometry()
      cil.framework.framework.AcquisitionGeometry

>>> A.norm()
      0.97807384
```

Under the hood `ProjectionOperator` uses either

- ASTRA https://www.astra-toolbox.com/
- TIGRE https://github.com/CERN/TIGRE

Demonstration data set



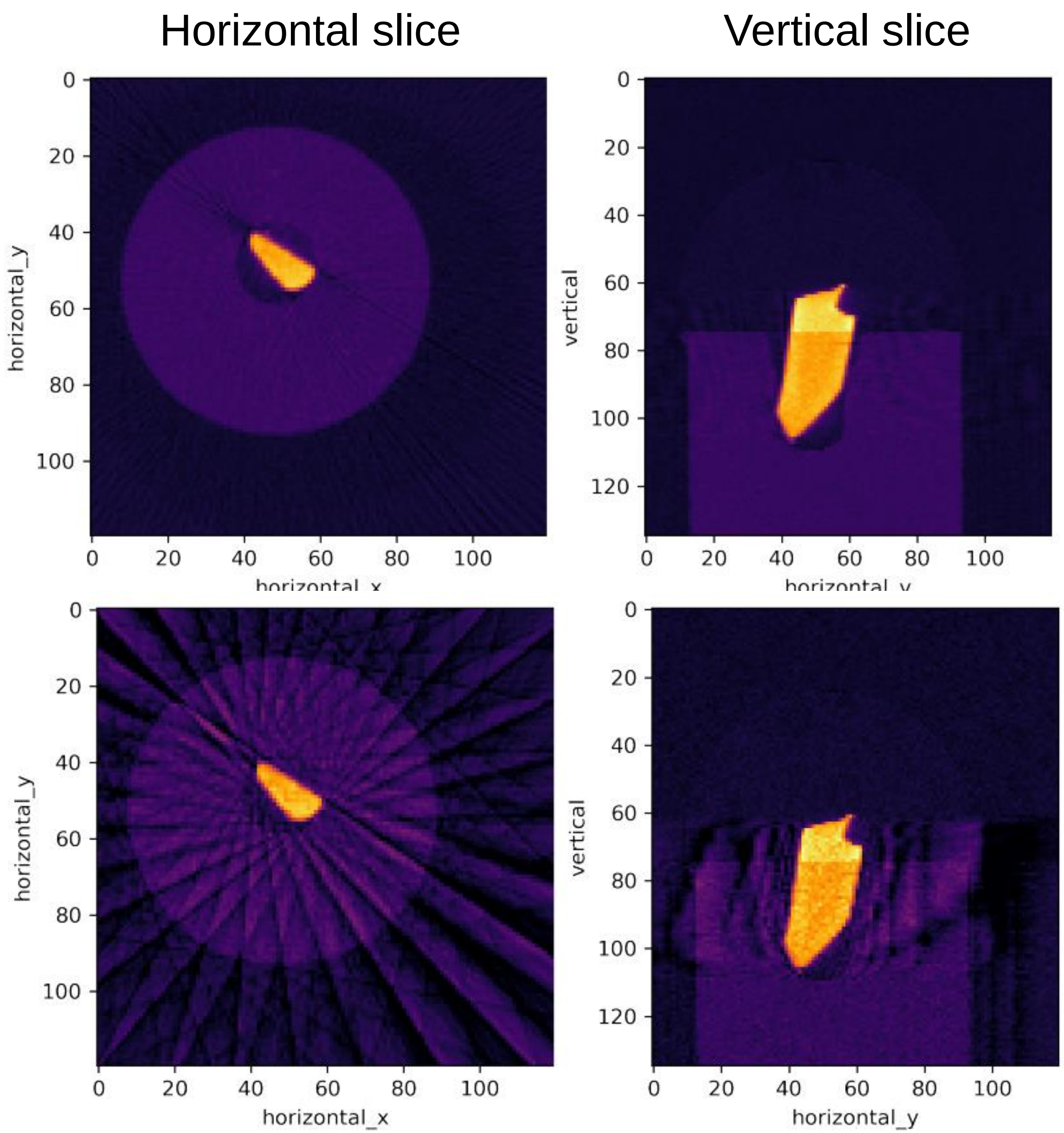
- 3D parallel-beam data set from Diamond Light Source, UK
- 0.5mm aluminium cylinder with piece of steel wire
- Droplet salt water causing corrosion + hydrogen bubbles
- Part of a fast time-lapse experiment
- 90 projections over 180 degrees, and **15 projections**
- Downsampled to 160-by-135 pixels for quick demonstration

J. et al. 2021: *Core Imaging Library - Part I: a versatile Python framework for tomographic imaging,* Phil Trans A. https://doi.org/10.1098/rsta.2020.0192

Filtered backprojection

90
projections

15
projections



Algebraic iterative methods (regularizing by number of iterations)

CGLS

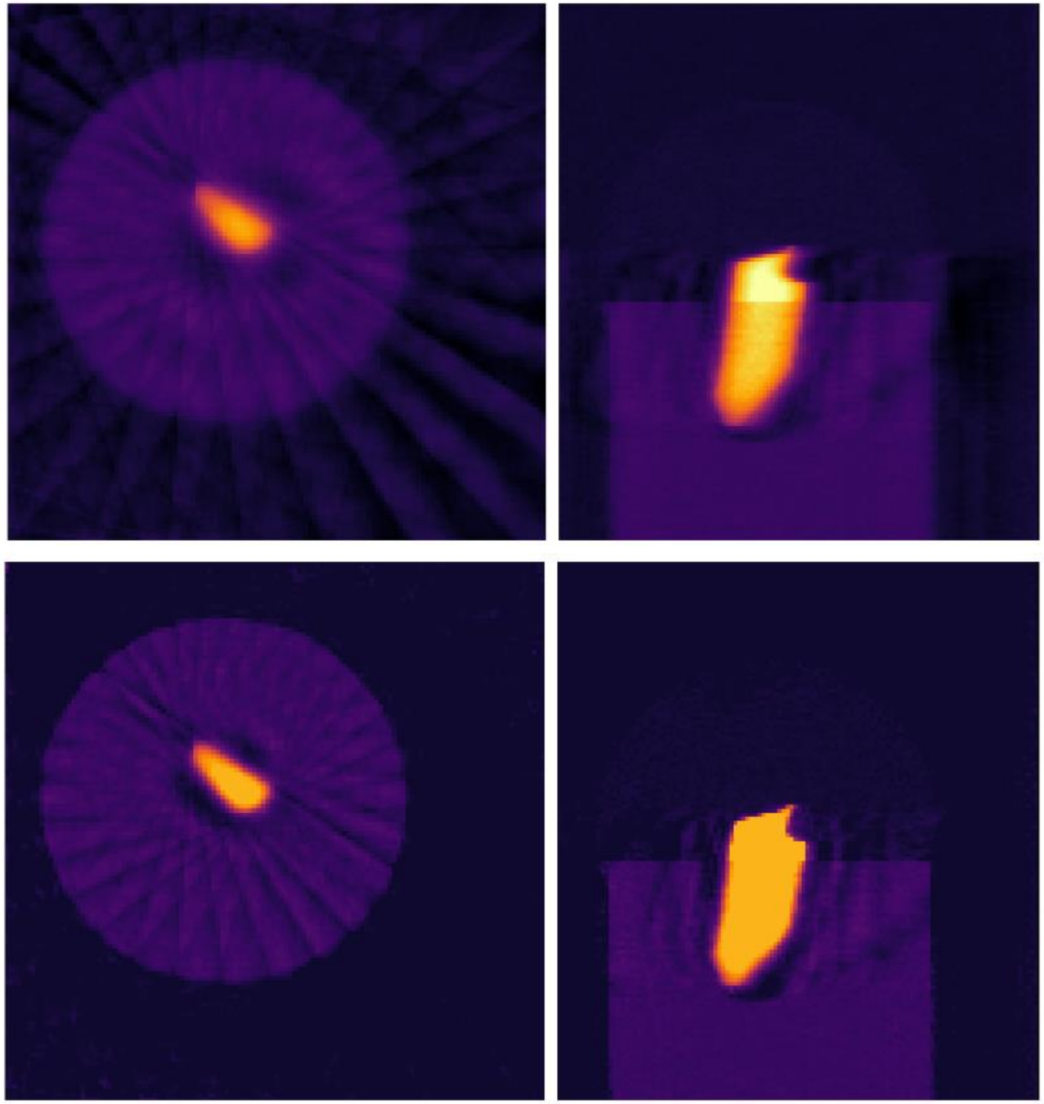
$$u^\star = \arg\min_u \|Au - b\|_2^2$$

Typically 10s of iterations

SIRT

As above and allowing lower and upper bounds on pixel values, here Non-negative and <= 0.9

Typically 100s of iterations



CGLS algorithm

How to set up and run an instance of the CGLS algorithm

```
initial = ig.allocate(0)

cgls = CGLS(initial=initial,
            operator=A,
            data=sino,
            max_iteration = 1000)

cgls.run(100)
```

More options and inputs available, see help.

Pause, do something, resume

```
cgls.run(3)

show2D(cgls.solution)

cgls.objective
      [1302.6678, 124.551254, 46.48474]

cgls.get_last_objective()
       46.48474

cgls.run(1)

cgls.objective
      [1302.6678, 124.551254, 46.48474, 25.61253]
```

SIRT algorithm

Similar to CGLS with additional lower/upper input for constraints:

```
sirt = SIRT(initial=x0,
            operator=A,
            data=sino,
            max_iteration=1000,
            lower=0.0,
            upper=17.0)

sirt.run(100)
```

More options and inputs available, see help.

Useful links for Core Imaging Library

- Website: https://www.ccpi.ac.uk/CIL
- Documentation: https://tomographicimaging.github.io/CIL
- Discord community discord.gg/9NTWu9MEGq

Software papers

- J. et al. 2021, Phil. Trans. R. Soc. A, **379**, 20200192:
  Core Imaging Library - Part I: a versatile Python framework for tomographic imaging
  https://doi.org/10.1098/rsta.2020.0192
- Papoutsellis et al. 2021, Phil. Trans. R. Soc. A, **379**, 20200193:
  Core Imaging Library - Part II: multichannel reconstruction for dynamic and spectral tomography
  https://doi.org/10.1098/rsta.2020.0193

Exercises

Exercise notebooks

• 04_FBP_CGLS_SIRT	Simulated data
• 05_usb_limited_angle_fbp_sirt	Real Zeiss cone-beam data

Exercises with own data

- Set up SIRT and CGLS for the central 2D slice of one of the data sets we acquired
- Explore the best number of iterations to run
- In SIRT, explore the use of upper and lower constraints
- Try 3D (extract 50-100 slices and/or crop data from the sides to reduce runtime)

Reference notebooks that may be useful to revisit

• 00_CIL_geometry	Overview of setting/modifying geometry
• 03_preprocessing	Prepreprocesing steps
• additional_exercises_data_resources	Ideas for further exploration

Option 1: Learnmore server

- Hands-on exercises at DTU Jupyter notebook server:
  https://learnmore1.compute.dtu.dk
  https://learnmore2.compute.dtu.dk

- To distribute load between servers:
  If your birth date is an odd (even) number, use learnmore1 (2)

- Use your DTU or guest login and password

- Assignments -> CINEMAXV_Reconstruction -> 2023_SC_for_CT_week_2
  -> Fetch -> Files -> Refresh (hit F5) -> Enter folder 2023_SC_for_CT_week_2

Option 2: STFC Cloud

- Hands-on exercises at the STFC Cloud:
  https://training.jupyter.stfc.ac.uk/

- 14 accounts with dedicated higher-spec GPU available – 2 accounts per group

• Group1:	sc4ct23_1	sc4ct23_2
• Group2:	sc4ct23_4	sc4ct23_5
• Group3:	sc4ct23_6	sc4ct23_7
• Group4:	sc4ct23_8	sc4ct23_9
• Group5:	sc4ct23_11	sc4ct23_12
• Group6:	sc4ct23_13	sc4ct23_14
• Group7:	sc4ct23_15	sc4ct23_16

Please note: accounts sc4ct23_3 & sc4ct23_10 are broken hence skipped above.

Option 2: STFC Cloud

1. Visit https://jupyter.stfc.ac.uk and click on sign up.
2. Enter your username (e.g. sc4ct23_1) and a password
   [note there is no password reset option – please remember it!]
3. Click login to return to the login page and again, enter the username and password.
4. Either you will be taken straight to the jupyter hub (if so, ignore this and the later steps) or you will be presented with server options.
5. Select **“OpenGL GPU environment”** and start
6. Starting the server may take some time.

- The exercise notebooks are in the folder
  CIL-Demos/demos/1-Introduction